Memory Allocation Diagnostic

Logging AllocP and FreeP calls

Tue, Feb 4, 2003

On more than a few occasions, the PowerPC system code has been beset by task suspension by the vxWorks kernel when a call to release a memory block is attempted via `free()`. This extreme reaction can quickly cause a front end to appear "dead." A recent example of this occurrence was a kind of "last straw" that prompted implementation of a diagnostic to keep track of memory allocations and look for unusual situations that occur when `FreeP` is called, and `free()` is about to be called. This note describes the initial implementation.

In keeping with many other system diagnostics, this one is designed around a data stream, which is here used to log calls to `AllocP` and `FreeP`. The 16-byte record format is as follows:

memory block address (4)
block size (2)
flags (1)
block type (1)
time-of-day (8)

The last byte of the normal BCD time-of-day is overwritten with the relative time within the current 15 Hz cycle in (binary) half ms units. The `flags` byte uses the hi nibble to indicate whether the record represents a call to `AllocP` (`0xA`) or `FreeP` (`0xF`). The low nibble of the `flags` byte is an error code detected by `FreeP`, with the following possible values:

0	No errors detected
1	Memory block already free
2	Invalid back ptr at end of block
3	Invalid previous ptr, size
4	n.u.

One problem with the diagnostic is that when a block is allocated, the block type is unknown, so the logged entry cannot show it; instead, it merely shows the initial value of `0x07`. One way to fix this is to add another argument to `AllocP` that would include the intended block type, but a cursory look at the source code made this seem rather daunting for now. Another approach is to take the block type that is known when the block is freed and try to search backward to find the matching allocation entry and patch it. We ignore this problem for now.

Implementation details

A low memory location was chosen (`ALLOCDSQ` at offset `0x69C`) to hold a pointer to the data stream queue, or `NULL`, if there is no data stream named `ALLOCLOG` defined in the `DSTRM` system table. The `ALLOCLOGInit` routine is called from `InzSys`, following the call to `DSInit`, to set up this low memory variable.

The new code was added to the `Alloc.c` module, including the `ALLOCLOGInit` routine above and the additions to `AllocP` and `FreeP`. In order to keep efficiency high, the usual call to `DSWrite` is emulated, an approach that has been used for several other diagnostics.

The main focus was the `FreeP` routine, which now includes several checks on the memory block to be freed before invoking the standard `free()` routine. If any of the three checks fails, then `free()` is not called. While this can result in a memory leak, it may not cause a system crash for a long while, during which time we may be able to find the bug that caused the check to fail. A caveat should be noted here. We think we understand the 8-byte vxWorks structure that we see around memory blocks that it allocates. But if our understanding is imperfect, this approach may be similarly imperfect. In addition, we must recognize that this logic is based upon the vxWorks kernel that we now use. Another kernel, or even a newer version of the vxWorks kernel, may allocate memory differently.

We understand the following about the 8-byte block structure that the vxWorks kernel uses to support its memory allocation scheme. It consists of two long words:

`prevBlk`	ptr to previous allocation structure
`thisSize`	size of the present allocated block, including these 8 bytes

The least bit of `thisSize` is set = 1 in the case that the allocated block is actually free memory; otherwise, the block has been allocated. By inspection, the following seems to be true:

1. The `thisSize` field in the current block structure, which immediately precedes the current memory block, is even if the current block is allocated.
2. Adding `thisSize` to the address of the current block structure gives the address of the structure just beyond the current block. The `prevBlk` field of that structure contains the address of the current block structure.
3. Using the current block structure `prevBlk` field to find the previous structure, then adding the `thisSize` field to that address results in the address of the current structure.

The new code in `FreeP`, besides logging the occurrence of the call to `FreeP`, makes the above three checks, in that order. The first one that fails means the logged entry will use that value 1–3 as the error code in the low nibble of the `flags` field. For any detected error, a 16-bit error count is incremented. The error counts are found in the user area of the data stream queue header. (There are normally at least 8 bytes of user area specified in a `DSTRM` entry.) Ideally, all such checks made in `FreeP` will succeed, and no error counts will be incremented. We need some experience with this diagnostic to verify how true this is. In a case that an array in a block has been overrun such that the following 8-byte block structure is corrupted, this diagnostic might give us a clue that could lead to discovering how it became corrupted. If we never see such counts, it gives added assurance that things are functioning properly. It may be useful to find a way to generate an alarm message if a check fails.

While making the above checks, if the 8-byte block structure is corrupted, a bus error may occur. To help forestall some of those possibilities, the new code only works with blocks that are less than 64K bytes in size. Use of `AllocP` to allocate a larger block can therefore result in an error encountered and a possible memory leak. It might be interesting to find out whether this circumstance occurs. Further checks could be made to insure that a `prevBlk` address does not stray too far afield, again to lessen the chance of getting a bus error when performing these checks.

Anyway, this first implementation is a start at monitoring memory allocation activity in a system. More features may be added in the future.

Refinements

The `ALLOCLOG` support that capture calls to `AllocP` and `FreeP` can be improved. This note explores some relatively simple ideas.

One drawback of the initial implementation is that the `AllocP` records do not indicate the memory block type being allocated. This is because it is unknown at the time that `AllocP` is invoked. But it is quite likely that the `mBlkType` field will be set in the block header very soon after the call to `AllocP`. The suggestion here is to check for it during the next time an entry is written into the data stream log and update the block type field that is found where the previous block address, assuming that the previous record is a record that allocates rather than one that frees.

To implement this suggestion, change the logic so that the `IN` offset to the queue is not advanced until the next entry is about to be written; *i.e.*, advance to the next record just before recording the current record. In this way, access to the previous record can be made for the purpose of updating the block type field just before moving on to record the present new record. A side effect of this is that the current record is not made officially a part of the queue until the next call to allocate or free memory is

made. If a user makes an access to the data stream records with a one-shot data request immediately after some condition is met, this should not cause a problem, because, in order to initialize the data request, an allocation will be made, so that by the time the data is collected for return, the last entry will be complete. If this suggestion is implemented, it may be a good idea to notice when the first record is written and realize that there is no previous record to check; therefore, the `IN` offset need not be advanced for the first record.

Another drawback is the chance that corruption of the 8-byte block structure could result in a bus error when a diagnostic record is written. To improve this situation, check more carefully the addresses obtained from that structure that might be used as pointers to access nearby structures. Limiting the size of an allocation to 64K bytes, as is already done, can help here. The `prevBlk` addresses in the block structures should not stray too far afield, because the block sizes are not too large. So, the address of the next block structure should not be more than 64K beyond the location of the current structure. The current logic guarantees this because it only looks at the low 16 bits of the block size field. Similarly, the `prevBlk` field of the current block structure should not be more than 64K before the current block structure.

There can be three new specific checks to come out of this:

a. The `blkSize` of the current block structure must be less than 64K.
b. The `blkSize` of the previous block structure must be less than 64K.
c. The `prevBlk` of the current block structure must be not too far back.

To implement these additional checks, we may want to change the assignment of error numbers in the logged entries used for `FreeP` calls. In addition, there must be more user area available in the `DSTRM` entry data stream definition. Right now, 8 bytes are to be available, which is enough room for 4 error counts. The next logical size to reserve is 24 bytes, which would allow for up to 12 error counts. An alternative would be to use only 8 bits for a count, since we do not expect to see any such counts. In this way, 8 error counts could be supported in 8 bytes.

Another drawback is that we have not yet implemented support for an alarm indication in case an error count is incremented. If the fact that a nonzero count is found is enough to make a channel or a bit bad, when would it be cleared? In order to annoy operators less in the event this takes place, and yet to alert the programmers, one might use a bit, since the original bit-based alarms are not directly supported by Acnet, but only if they are part of a "combined binary" status word. The programmer might clear the counts in order to clear the alarm condition. The condition would be logged in the private Linac log that does not impact the operator alarm screen. It would not be at all secret.

Post-refinement version

With the additions made to the diagnostic code described above, this is the latest situation. The error codes detected by `FreeP` and installed in the low 4 bits of the `flags` byte, have been reassigned, so that they now have the following meanings:

0. No error.
1. The `blkSize` field in the current block structure is >= 64K.
2. The `prevBlk` field in the current block structure is more than 64K back.
3. The `blkSize` field in the previous block structure is >= 64K.
4. The current block is already free.
5. The `prevBlk` field in the next block structure does not point to the current structure.
6. The `blkSize` field in the previous block structure, does not lead to the current block.

If any of these errors is detected, a count is incremented. the counts are now only 8 bits wide, so that as many as 8 error codes can be supported without increasing the size of the user area in the data stream queue header. (That size is required to allow for 8 bytes to provide space for the count bytes.) The counts cannot increase past 255. A total count is provided as a 16-bit value in the `SPAR1` field of

the data stream queue type 1 header, which is located immediately after the `START` field. Monitoring this value, copying it to an analog channel, for being nonzero can be used to cause an alarm message.

The code now follows the first suggested refinement so that the block type field in the `AllocP` logged entry is updated when the next logged entry is created. Here is an example of the list of logged records soon after `node0590` was reset:

```
FILE<0590>             01/31/03 0942
 0590:00EF4000   0001 0010 0018 1000 0000 002F 0000 0000  TOTAL records written = 47
     :00EF4010   0300 1000 0020 0000 0000 0000 0000 0000  IN offset = 0x300, no error counts
     :00EF4020   0097 AD88 0048 A009 0301 3109 4123 0662  Alloc Acnet message block
     :00EF4030   0097 A980 0400 A007 0301 3109 4123 060B  Alloc DNSQ LA context
     :00EF4040   0097 8B10 0480 A007 0301 3109 4123 0611  Alloc TFTP LA context
     :00EF4050   0097 A690 0060 A015 0301 3109 4123 134D  Alloc port# block
     :00EF4060   0093 BF50 16E8 A007 0301 3109 4123 131B  Alloc AAUX LA context
     :00EF4070   0093 FC10 0060 A015 0301 3109 4123 142C  Alloc port# block
     :00EF4080   0093 FB78 0090 A002 0301 3109 4123 1405  Alloc type#2 request block
     :00EF4090   0093 F830 0110 A007 0301 3109 4124 0005  Alloc SLOG LA context
     :00EF40A0   00DF F2E8 0060 A015 0301 3109 4124 026F  Alloc port# block
     :00EF40B0   00DF F2B0 0030 A001 0301 3109 4124 0270  Alloc type#1 request block (SLOG)
     :00EF40C0   00DF E9D8 0688 A007 0301 3109 4124 0270  Alloc DBDL LA context
     :00EF40D0   00DF B2A8 0800 A007 0301 3109 4124 0581  Alloc REQM LA context
     :00EF40E0   00DF E928 0030 A001 0301 3109 4124 0526  Alloc type#1 request block (REQM)
     :00EF40F0   0093 91F0 01D0 A007 0301 3109 4124 0526  Alloc GATE LA context
     :00EF4100   0097 CF80 2458 A007 0301 3109 4124 0532  Alloc ECHO LA context (UDP Echo se
     :00EF4110   0097 C1D0 00C8 A007 0301 3109 4124 073A  Alloc KRFP LA context
     :00EF4120   0097 B418 0060 A015 0301 3109 4124 0910  Alloc port# block
     :00EF4130   0097 C128 00A0 A00A 0301 3109 4125 0635  Alloc comment alarm support block
     :00EF4140   0092 FB00 0820 A00A 0301 3109 4125 060D  Alloc analog alarm support block
     :00EF4150   0092 F2D8 0820 A00A 0301 3109 4125 060D  Alloc binary alarm support block
     :00EF4160   0097 AD88 0048 F009 0301 3109 4125 070D  Free Acnet message block
     :00EF4170   0097 AD98 0038 A004 0301 3109 4125 0704  Alloc alarm message block
     :00EF4180   0097 C0C0 0060 A015 0301 3109 4125 0704  Alloc port# block
     :00EF4190   0097 C080 0038 A004 0301 3109 4125 0704  Alloc alarm message block
     :00EF41A0   0097 AD98 0038 F004 0301 3109 4125 0705  Free alarm message block
     :00EF41B0   0097 C080 0038 F004 0301 3109 4125 0705  Free alarm message block
     :00EF41C0   0093 FB78 0090 F002 0301 3109 4136 0305  Free type#2 request block
     :00EF41D0   0093 FBA8 0060 A015 0301 3109 4137 010A  Alloc port# block
     :00EF41E0   0097 AD98 0038 A002 0301 3109 4137 0126  Alloc type#2 request block
     :00EF41F0   0097 AD98 0038 F002 0301 3109 4137 0126  Free type#2 request block
     :00EF4200   0097 C058 0060 A015 0301 3109 4137 0126  Alloc port# block
     :00EF4210   0097 AD98 0038 A002 0301 3109 4137 0137  Series of similar one-shot name lo
     :00EF4220   0097 AD98 0038 F002 0301 3109 4137 0137  about 4 ms apart.
     :00EF4230   0097 AD98 0038 A002 0301 3109 4137 0137
     :00EF4240   0097 AD98 0038 F002 0301 3109 4137 013F
     :00EF4250   0097 AD98 0038 A002 0301 3109 4137 013F
     :00EF4260   0097 AD98 0038 F002 0301 3109 4137 0147
     :00EF4270   0097 AD98 0038 A002 0301 3109 4137 0147
     :00EF4280   0097 AD98 0038 F002 0301 3109 4137 0150
     :00EF4290   0097 AD98 0038 A002 0301 3109 4137 0150
     :00EF42A0   0097 AD98 0038 F002 0301 3109 4137 0159
     :00EF42B0   0097 AD98 0038 A002 0301 3109 4137 0159
     :00EF42C0   0097 AD98 0038 F002 0301 3109 4137 0162
     :00EF42D0   0097 AD98 0038 A002 0301 3109 4137 0162
     :00EF42E0   0097 AD98 0038 F002 0301 3109 4137 0169
     :00EF42F0   0097 BFF0 0060 A015 0301 3109 4137 1369  Alloc port# block
     :00EF4300   0092 EE98 0438 A007 0301 3109 4145 1364  Alloc block, type as yet unknown.
     :00EF4310   0000 0000 0000 0000 0000 0000 0000 0000  (no more entries yet written)
```

Emergency addendum

When the new code was installed in the Linac front ends, a problem showed up. As stated in the refinements section of this note, the logic assumed that the previous memory block would be limited in size, meaning it should be less than 64K bytes. But this assumes that the previous block is one that we allocated. But it could be that it was allocated for a different purpose under vxWorks, in which case we cannot assume it is small. To allow for this, that part of the checking was removed, so that this is the final (latest) modified list of error meanings:

0. No error.
1. The `blkSize` field in the current block structure is >= 64K.
2. The current block to be freed is already free.
3. The `prevBlk` field in the next block structure does not point to the current structure.
4. The `blkSize` field in the previous block structure does not lead to the current block.

Perhaps the logic may be made a bit more robust in the future to forestall bus errors that may occur if the memory block structures are corrupted. But this will have to do for now.